Appendix 1: Source Code for Sample Implementation of Access Delay Reduction Algorithm

This section contains sample C++ source code for a floating-point version of the Access Delay Reduction algorithm. 5 files are listed.

File	Description
pseudocode.c	Pseudo C++ code that shows how to call the ADR algorithm in an application. No implementations are given for many of the functions called in this code as they are system dependent.
adr.h	Header file for the Access Delay Reduction algorithm. Includes declarations for both the public and private parts of the class. Internally, this class uses the CircularBuffer class.
adr.c	Implementation of the Access Delay Reduction algorithm. This file contains the heart of the algorithm and is the most important file included here.
circularbuffer.h	Include file for a circular First In First Out (FIFO) buffer. The CircularBuffer is used internally by adr.c. It is not called directly by the user and is included to clarify its use by the AccessDelayReduducer class.
circularbuffer.c	Implementation of the circular buffer. This file includes "*libcoder.h*" which is not shown here. The only function declared in *libcoder.h* is the *error* function, which halts the system on catastrophic errors.

File: pseudocode.c

```
 1 /*
 2  * Copyright (c) 1999-2000 AT&T Corp.
 3  * All Rights Reserved.
 4  */
 5 #include <circularbuffer.h>
 6 #include <vad.h>
 7 #include <adr.h>

 8 /*
 9  * pseudo code for main processing loop with Access Delay Reduction algorithm.
10  * Read a frame's worth of audio, give it to both the VAD and ADR. When
11  * the VAD detects onset of activity, request a transmission channel. In
12  * the mean time the ADR buffers the speech. After the access delay, the
13  * ADR time-scales the beginning of the talkspurt until the access delay
14  * is gone. At the end of the talkspurt, the transmit channel is freed.
15  */
16 void processloop()
17 {
18    int                framesz = 160;      /* 20 msec at 8 KHz */
19    Float              y[160];
20    bool               activity, oldactivity = false;
21    bool               adrdata, oldadrdata = false;
```

```
22    Vad                   vad(8000, 160);
23    AccessDelayReducer    adr(8000, 20., 60., 500.);

24    while (readinputframe(y, framesz)) {
25          activity = vad.activity(y);
26          /* request transmission channel at activity onset */
27          if (activity && !oldactivity)
28                request_tx_channel();
29          adrdata = adr.newframe(y, y, activity);
30          if (adrdata)
31                encode_and_xmit(y, framesz);
32          /* free channel when ADR buffer has drained */
33          if (!adrdata && oldadrdata)
34                free_tx_channel();
35          oldactivity = activity;
36          oldadrdata = adrdata;
37    }
38 }
```

Pseudocode.c

The function *processloop* in pseudocode.c shows how the AccessDelayReducer class is used in an application. Here, we have decided to process the speech in increments of 160 samples, or 20 msec at 8 KHz sampling. On line 19 an array large enough to hold one frame's worth of floats is declared. The "Float" type is defined as a float with a *typedef* in the file circularbuffer.h. The *bools* on lines 20 and 21 keep track of the current and previous state of both the VAD and the ADR. An inactive to active transition detected by the VAD is used to request a transmission channel on lines 27 and 28. On lines 33-34, the end of available data for a talkspurt is used to relinquish the transmission channel. The constructor for the VAD on line 22 sets the VAD frame size to 160 samples and the samplerate to 8 KHz. The constructor call to the AccessDelayReducer on line 23 sets the samplerate to 8 KHz, the frame size to 20 msec, the access delay to 60 msec, and the interval for the time-scaling to 500 msec.

The loop on lines 24-37 reads in a frame of data and processes it. First, the VAD determines if there is activity on line 25. Next the frame is given to the ADR on line 29. The first argument is the input frame and the second argument is the output frame. In this example, the output from the ADR is placed in the same buffer used for input. The speech is buffered and delayed internally by the ADR. The call to *newframe* returns true if the output frame contains speech that should be transmitted (there is activity in it) and false otherwise. At the first few frames after an inactive to active transition in the VAD, e.g. for the duration of the access delay, *newframe* returns false even though the input frames contain active speech. After the access delay is over, the speech at the start of the talkspurt is returned. *Newframe* then starts time-scale compressing the speech until the access delay is removed.

Since the ADR may leave some residual delay or the talkspurt may be too short for the ADR to finish time-scaling, the output of the ADR determines when the transmission channel is returned rather than the VAD. All the active speech buffered in the ADR must be output before channel is returned.

File: adr.h

```
  1 /*
```

```
 2  * Copyright (c) 1999-2000 AT&T Corp.
 3  * All Rights Reserved.
 4  *
 5  * Performing time-scaling compression at the start of a talkspurt
 6  * in systems where there is access delay for channel allocation such
 7  * as Voice over EDGE.
 8  */
 9 class AccessDelayReducer {
10 public:
11          AccessDelayReducer(int srate, Float framesizems,
12                Float accessdelayms, Float timescaleintervalms);
13          ~AccessDelayReducer();
14    bool   newframe(Float *in, Float *out, bool active);
15 protected:
16    Float  frameszmsec;  /* frame size in msec */
17    Float  sysdelaymsec; /* system contention delay, msec */
18    Float  timescalemsec;      /* interval for timescaling, msec */
19    Float  targetaccum;  /* target accumulator, samples */
20    Float  targetincr;   /* target increment, samples */
21    int    samplerate;   /* samplerate, Hz */
22    int    framesz;      /* frame size in samples */
23    int    activelen;    /* frames in current talk spurt */
24    int    sysdelayf;    /* system contention delay, frames */
25    int    sysdelay;     /* system delay, samples */
26    int    curdelay;     /* current delay, samples */
27    int    targetdelay;  /* target delay, samples */
28    int    timescalef;   /* timescaling interval, frames */
29    int    timescalefirstf;/* first frame to start timescaling */
30    int    timescalelastf;     /* last frame to start timescaling */
31    int    ndec;         /* decimation factor */
32    int    pitchmin;     /* minimum pitch */
33    int    pitchmax;     /* maximum pitch */
34    int    pitchdiff;    /* pitch difference */
35    int    corrlen;      /* correlation length */
36    int    corrbuflen;   /* length of correlation buffer */
37    CircularBuffer       *outbuf;/* output buffer */
38    Float  *tmpbuf;      /* temporary scratch buffer */
39    Float  *corrbuf;     /* input buffer */

40    int    findbestmatch();
41    void   updatecorrbuf(Float *s);
42    void   removedelay(Float *in, int pitch);
43    void   overlapadd(Float *l, Float *r, Float *o, int cnt);
44    void   idle();
45    void   copy(Float *f, Float *t, int cnt);
46    void   zero(Float *s, int cnt);
47 };
```

File: adr.c

```
 1 /*
 2  * Copyright (c) 1999-2000 AT&T Corp.
 3  * All Rights Reserved.
 4  */
 5 #include <math.h>
 6 #include "circularbuffer.h"
 7 #include "adr.h"

 8 #define  PITCH_MIN     .0025          /* minimum allowed pitch, 400 Hz */
 9 #define  PITCH_MAX     .015           /* maximum allowed pitch, 66 Hz */
10 #define  NDEC_8K       2              /* 2:1 decimation at 8kHz */
11 #define  CORRMINPOWER  ((Float)250.)  /* minimum power */
12 #define  CORRLEN       .020           /* 20 msec correlation length */
```

```
13 /*
14  * Constructor sets the samplerate, the frame size, the estimated access delay
15  * and the time-scaling interval. Appropriate length buffers are allocated
16  * based on these parameters.
17  */
18 AccessDelayReducer::AccessDelayReducer(int srate, Float framesizems,
19    Float accessdelayms, Float timescaleintervalms)
20 {
21    samplerate = srate;
22    frameszmsec = framesizems;
23    sysdelaymsec = accessdelayms;
24    timescalemsec = timescaleintervalms;
25    ndec = (int)(NDEC_8K * samplerate / 8000.);
26    pitchmin = (int)(PITCH_MIN * samplerate);
27    pitchmax = (int)(PITCH_MAX * samplerate);
28    pitchdiff = pitchmax - pitchmin;
29    corrlen = (int)(CORRLEN * samplerate);
30    corrbuflen = corrlen + pitchmax;
31    framesz = (int)(samplerate * frameszmsec * (Float).001);
32    sysdelayf = (int)ceil(sysdelaymsec / frameszmsec);
33    sysdelay = sysdelayf * framesz;
34    timescalef = (int)ceil(timescalemsec / frameszmsec) + 1;
35    timescalefirstf = sysdelayf + 1;
36    timescalelastf = sysdelayf + timescalef;
37    targetincr = (Float)sysdelay / (timescalef + 1);
38    corrbuf = new Float[corrbuflen];
39    outbuf = new CircularBuffer(framesz * (sysdelayf + 1));
40    tmpbuf = new Float[pitchmax >> 2];
41    activelen = 0;
42    idle();
43 }

44 /*
45  * Free allocated resources in destructor.
46  */
47 AccessDelayReducer::~AccessDelayReducer()
48 {
49    delete [] tmpbuf;
50    delete outbuf;
51    delete [] corrbuf;
52 }

53 /*
54  * main public function for time-scaling sppech at start of talkspurt.
55  * Input is the speech from the audio port and active indicator from the
56  * VAD. Output is the speech delayed by the access delay, and then time-scaled
57  * to get remove the delay at the start of the talksprt.
58  * Newframe returns true if the returned frame should be transmitted and
59  * false if it should not be transmitted. For simulation purposes the
60  * returned frame of speech is set to zero if it should not be transmitted.
61  */
62 bool AccessDelayReducer::newframe(Float *in, Float *out, bool active)
63 {
64    bool   r;
65    int    pitch, cnt;

66    updatecorrbuf(in);
67    if (active) {
68          /* simulate contention delay at start of utterance */
69          if (++activelen <= sysdelayf) {
70                /*
71                 * if delayed samples still left from last utterance
72                 * flush it. This shouldn't happen since if there
```

```
73                  * is some leftover delay, it should be output at
74                  * the first frame where the VAD determines there is
75                  * no activity.
76                  */
77                 if (activelen == 1 && outbuf->filled())
78                       outbuf->flush();
79                 outbuf->write(in, framesz);
80                 curdelay += framesz;
81                 zero(out, framesz);
82                 r = false;
83           }
84           /* time-scale at start of utterance */
85           else {
86                 /* update the current amount we allow to timescale */
87                 if (activelen <= timescalelastf) {
88                       /*
89                        * boost at first frame so targetaccum is
90                        * greater than pitchmin so its possible
91                        * to time-scale at frame timescalefirstf.
92                        */
93                       if (activelen == timescalefirstf)
94                             targetaccum = (Float)2. * targetincr;
95                       else
96                             targetaccum += targetincr;
97                       targetdelay = (int)targetaccum;
98                       if (targetdelay > curdelay)
99                             targetdelay = curdelay;
100                }
101                /*
102                 * if the target for delay removal is larger than
103                 * the minimum pitch, we can try to remove the delay.
104                 * We still may not be able to do it yet if the
105                 * estimated pitch is larger than the target delay.
106                 */
107                if (targetdelay >= pitchmin &&
108                    (pitch = findbestmatch()) <= targetdelay) {
109                      removedelay(in, pitch);
110                      outbuf->read(out, framesz);
111                }
112                /*
113                 * either time-scaling isn't necessary, or not
114                 * possible because not enough time has passed,
115                 * or the current pitch is too long.
116                 * If outcnt is 0, all the delay has been removed
117                 * so we just copy the data from input to output.
118                 * Otherwise, there is still delay in the system
119                 * so the output must be buffered.
120                 */
121                else if (outbuf->filled() == 0)
122                      copy(in, out, framesz);
123                else {
124                      outbuf->write(in, framesz);
125                      outbuf->read(out, framesz);
126                }
127                r = true;
128          }
129    }
130    /* no speech activity detected */
131    else {
132          if (activelen != 0) {
133                activelen = 0;
134                idle();
135          }
```

```
136           /* if something left in delay buffer, output it */
137           cnt = outbuf->filled();
138           if (cnt) {
139                 if (cnt >= framesz)
140                       cnt = framesz;
141                 int left = framesz - cnt;
142                 outbuf->read(out, cnt);
143                 zero(&out[cnt], left);
144                 if (outbuf->filled() == 0)
145                       idle();
146                 r = true;
147           } else {
148                 zero(out, framesz);
149                 r = false;
150           }
151     }
152     return r;
153 }

154 /* remove the delay by time-scale compressing the input */
155 void AccessDelayReducer::removedelay(Float *in, int pitch)
156 {
157     int    p2, pq, cnt, olacnt, ocnt;

158     /* see if we can remove more than one pitch period at a time */
159     p2 = pitch << 1;
160     if (p2 <= targetdelay && p2 <= pitchmax)
161           pitch = p2;
162     pq = pitch >> 2;
163     olacnt = pitch + pq;
164     /* if the OLA fits in one frame, work directly on the input frame */
165     if (olacnt <= framesz) {
166           cnt = framesz - olacnt;
167           outbuf->write(in, cnt);
168           overlapadd(&in[cnt], &in[cnt+pitch], tmpbuf, pq);
169           outbuf->write(tmpbuf, pq);
170     }
171     /* Otherwise we have to copy some samples from the previous frame */
172     else {
173           cnt = olacnt - framesz;
174           ocnt = pq - cnt;
175           outbuf->peektail(tmpbuf, cnt);    /* from previous frame tail */
176           copy(in, &tmpbuf[cnt], ocnt);     /* from current frame */
177           overlapadd(tmpbuf, &in[framesz - pq], tmpbuf, pq);
178           outbuf->replacetail(tmpbuf, cnt); /* replace old tail */
179           outbuf->write(tmpbuf, ocnt);      /* write tail of OLA */
180     }
181     /* update the current delay variables */
182     targetaccum -= (Float)pitch;
183     targetdelay -= pitch;
184     curdelay -= pitch;
185 }

186 /* Initialized the time-scaling variables */
187 void AccessDelayReducer::idle()
188 {
189     curdelay = 0;
190     targetdelay = 0;
191     targetaccum = 0.;
192 }

193 /* Save a frames worth of new speech into the correlation buffer */
194 void AccessDelayReducer::updatecorrbuf(Float *s)
```

```
195 {
196    int offset = corrbuflen - framesz;
197    /* make room for new speech frame */
198    copy(&corrbuf[corrbuflen - offset], corrbuf, offset);
199    /* copy in the new frame */
200    copy(s, &corrbuf[offset], framesz);
201 }

202 /*
203  * Find the best match between the last segment of speech and
204  * the previous speech in the correlation buffer.
205  */
206 int AccessDelayReducer::findbestmatch()
207 {
208    int    i, j, k;
209    int    bestmatch;
210    Float  bestcorr;
211    Float  corr;          /* correlation */
212    Float  energy;              /* running energy */
213    Float  scale;         /* scale correlation by average power */
214    Float  *rp;           /* segment to match */
215    Float  *l;

216    l = &corrbuf[pitchmax];
217    /* coarse search */
218    rp = corrbuf;
219    energy = 0.f;
220    corr = 0.f;
221    for (i = 0; i < corrlen; i += ndec) {
222           energy += rp[i] * rp[i];
223           corr += rp[i] * l[i];
224    }
225    scale = energy;
226    if (scale < CORRMINPOWER)
227           scale = CORRMINPOWER;
228    corr /= (Float)sqrt(scale);
229    bestcorr = corr;
230    bestmatch = 0;
231    for (j = ndec; j <= pitchdiff; j += ndec) {
232           energy -= rp[0] * rp[0];
233           energy += rp[corrlen] * rp[corrlen];
234           rp += ndec;
235           corr = 0.f;
236           for (i = 0; i < corrlen; i += ndec)
237                  corr += rp[i] * l[i];
238           scale = energy;
239           if (scale < CORRMINPOWER)
240                  scale = CORRMINPOWER;
241           corr /= (Float)sqrt(scale);
242           if (corr >= bestcorr) {
243                  bestcorr = corr;
244                  bestmatch = j;
245           }
246    }
247    /* fine search */
248    j = bestmatch - (ndec - 1);
249    if (j < 0)
250           j = 0;
251    k = bestmatch + (ndec - 1);
252    if (k > pitchdiff)
253           k = pitchdiff;
254    rp = &corrbuf[j];
255    energy = 0.f;
```

```
256     corr = 0.f;
257     for (i = 0; i < corrlen; i++) {
258             energy += rp[i] * rp[i];
259             corr += rp[i] * l[i];
260     }
261     scale = energy;
262     if (scale < CORRMINPOWER)
263             scale = CORRMINPOWER;
264     corr = corr / (Float)sqrt(scale);
265     bestcorr = corr;
266     bestmatch = j;
267     for (j++; j <= k; j++) {
268             energy -= rp[0] * rp[0];
269             energy += rp[corrlen] * rp[corrlen];
270             rp++;
271             corr = 0.f;
272             for (i = 0; i < corrlen; i++)
273                     corr += rp[i] * l[i];
274             scale = energy;
275             if (scale < CORRMINPOWER)
276                     scale = CORRMINPOWER;
277             corr /= (Float)sqrt(scale);
278             if (corr > bestcorr) {
279                     bestcorr = corr;
280                     bestmatch = j;
281             }
282     }
283     return pitchmax - bestmatch;
284 }

285 /* Overlap add with triangular windows */
286 void AccessDelayReducer::overlapadd(Float *l, Float *r, Float *o, int cnt)
287 {
288     Float incr = (Float)1. / cnt;
289     Float lw = (Float)1. - incr;
290     Float rw = incr;
291     for (int i = 0; i < cnt; i++) {
292             o[i] = lw * l[i] + rw * r[i];
293             lw -= incr;
294             rw += incr;
295     }
296 }

297 void AccessDelayReducer::copy(Float *f, Float *t, int cnt)
298 {
299     for (int i = 0; i < cnt; i++)
300             t[i] = f[i];
301 }

302 void AccessDelayReducer::zero(Float *s, int cnt)
303 {
304     for (int i = 0; i < cnt; i++)
305             s[i] = (Float)0.;
306 }
```

File: circularbuffer.h

```
  1 /*
  2  * Copyright (c) 1999-2000 AT&T Corp.
  3  * All Rights Reserved.
  4  *
  5  * Circular buffer
  6  */
```

```
 7 typedef float Float;

 8 class CircularBuffer {
 9 public:
10          CircularBuffer(int sz);
11          ~CircularBuffer();
12    void  read(Float *f, int sz);
13    void  write(Float *f, int sz);
14    void  peekhead(Float *f, int sz);
15    void  peektail(Float *f, int sz);
16    void  replacehead(Float *f, int sz);
17    void  replacetail(Float *f, int sz);
18    void  flush();
19    void  clear();
20    int   capacity()    { return buflen; }
21    int   filled()      { return cnt; }
22 protected:
23    int   buflen;             /* buffer size */
24    int   cnt;          /* valid samples in buffer */
25    Float *buf;         /* buffer */
26    Float *bufe;        /* buffer end */
27    Float *bufr;        /* buffer read pointer */
28    Float *bufw;        /* buffer write pointer */
29    void  copy(Float *f, Float *t, int cnt);
30 };
```

File: circularbuffer.c

```
 1 /*
 2  * Copyright (c) 1999-2000 AT&T Corp.
 3  * All Rights Reserved.
 4  */
 5 #include "libcoder.h"
 6 #include "circularbuffer.h"

 7 CircularBuffer::CircularBuffer(int sz)
 8 {
 9    buflen = sz;
10    buf = new Float[buflen];
11    bufe = &buf[buflen];
12    flush();
13 }

14 CircularBuffer::~CircularBuffer()
15 {
16    delete [] buf;
17 }

18 /* flush all data from the buffer */
19 void CircularBuffer::flush()
20 {
21    bufr = bufw = buf;
22    cnt = 0;
23 }

24 /* fill the buffer with all zeros */
25 void CircularBuffer::clear()
26 {
27    int   i;

28    bufr = bufw = buf;
29    cnt = buflen;
30    for (i = 0; i < buflen; i++)
31          buf[i] = 0.;
```

```
}

/*
 * Save data in the buffer. Its legal to write more data to the buffer
 * than it can hold. In this case just the latest data is kept and the
 * read pointer is updated.
 */
void CircularBuffer::write(Float *f, int sz)
{
    int left;

    cnt += sz;
    do {
            left = bufe - bufw;
            if (left > sz)
                    left = sz;
            copy(f, bufw, left);
            bufw += left;
            if (bufw == bufe)
                    bufw = buf;
            sz -= left;
            f += left;
    } while (sz);
    /*
     * if more data has been written than can fit,
     * update the read pointer so it reads the latest data.
     */
    if (cnt > buflen) {
            cnt = buflen;
            bufr = bufw;
    }
}

/* retrieve data from the buffer */
void CircularBuffer::read(Float *f, int sz)
{
    if (sz > cnt)
            ::error("CircularBuffer::read: read too large");
    cnt -= sz;
    int c = bufe - bufr;
    if (sz < c) {
            copy(bufr, f, sz);
            bufr += sz;
    } else {
            int c2 = sz - c;
            copy(bufr, f, c);
            copy(buf, &f[c], c2);
            bufr = &buf[c2];
    }
}

/*
 * return the first sz samples at the head of
 * the buffer without modifying the buffer
 */
void CircularBuffer::peekhead(Float *f, int sz)
{
    if (sz > cnt)
            ::error("CircularBuffer::peekhead: not enough data");
    int c = bufe - bufr;
    if (sz <= c)
            copy(bufr, f, sz);
    else {
```

```
91          copy(bufr, f, c);
92          copy(buf, &f[c], sz - c);
93      }
94  }

95  /* replace the first sz samples at the head of the buffer */
96  void CircularBuffer::replacehead(Float *f, int sz)
97  {
98      if (sz > cnt)
99          ::error("CircularBuffer::replacehead: not enough data");
100     int c = bufe - bufr;
101     if (sz <= c)
102         copy(f, bufr, sz);
103     else {
104         copy(f, bufr, c);
105         copy(&f[c], buf, sz - c);
106     }
107 }

108 /*
109  * return the last sz samples in the tail of
110  * the buffer without modifying the buffer
111  */
112 void CircularBuffer::peektail(Float *f, int sz)
113 {
114     if (sz > cnt)
115         ::error("CircularBuffer::peektail: not enough data");
116     int fromstart = bufw - buf;
117     if (sz > fromstart) {
118         int c = sz - fromstart;
119         copy(bufe - c, f, c);
120         f += c;
121         sz -= c;
122     }
123     copy(bufw - sz, f, sz);
124 }

125 /* replace the last sz samples in the tail of the buffer */
126 void CircularBuffer::replacetail(Float *f, int sz)
127 {
128     if (sz > cnt)
129         ::error("CircularBuffer::replacetail: not enough data");
130     int fromstart = bufw - buf;
131     if (sz > fromstart) {
132         int c = sz - fromstart;
133         copy(f, bufe - c, c);
134         f += c;
135         sz -= c;
136     }
137     copy(f, bufw - sz, sz);
138 }

139 void CircularBuffer::copy(Float *f, Float *t, int cnt)
140 {
141     for (int i = 0; i < cnt; i++)
142         t[i] = f[i];
143 }
```